Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 3

Hopfield Networks

Rahul Shah, Nicolas Essipova, and Love Marcus

February 15, 2019

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to implement a functional Hopfield network,
- to get some practical experience of auto-associative networks, and
- to examine the capacity and robustness of the Hopfield network.

2 Methods

We have implemented the Hopfield network in python with Jupyter Notebook. We have worked with numpy and the figures are generated with matplotlib. Permutations were generated with itertools.

3 Results and discussion

3.1 Convergence and attractors

All the three patterns converged with the asynchronous update rule. There are 6 attractors with the asynchronous update rule of the weights on the diagonal axis (w_{ii}) are set to zero. 10 otherwise.

The 6 attractors stem from the provided patterns and their respective anti-patterns (inverted patterns).

We tested the Hopfield network with noisy inputs. The recall is sometimes correct when there are 3 incorrect bites and very seldom with 4.

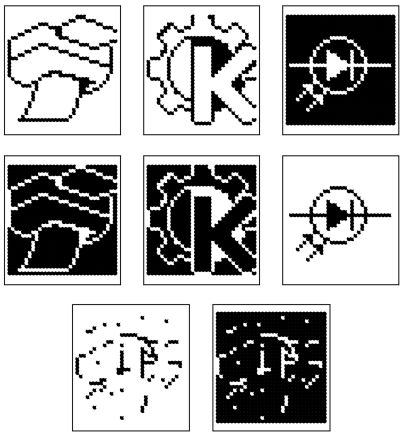
3.2 Sequential update

Our guess here is that sequential update is synonymous with asynchronous updates.

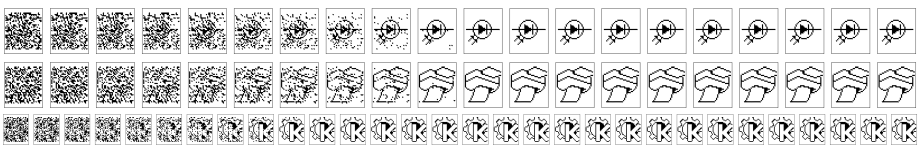
The three provided patterns are stable and so are the anti-patterns. We also observed a spurious pattern as in Figure 1. In this text we refer to the negative patterns as anti-patterns and the combined patterns as spurious.

The network successfully recreates p1 from p10. p11 sometimes leads to p2, sometimes to p3 and sometimes to the spurious pattern. The spurious pattern is a combination between the patterns that the network was trained with.

Any and images in Figure 1 is recalled by the network when random noise is submitted. Examples of the process is provided in Figure 2. Figure 2 show snapshots of the process every 100 node updates. Each node is only updated a maximum of 2 times for the network to stabilize.



Figur 1: The provided patterns at the top and their anti-patterns below. The bottom images display a spurious pattern that appeared.

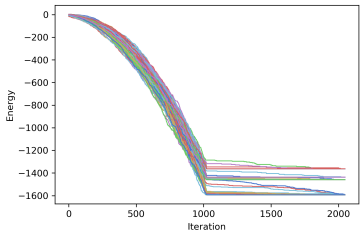


Figur 2: Intermediate states from every 100th node update.

3.3 Energy

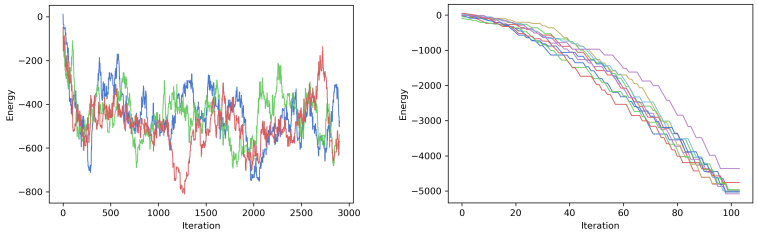
The energies of the patterns are (-1436.4, -1362.6, -1459.3) in the same order as they appear in Figure 1. The energies are the same for the anti-patterns due to the symmetry. The energy of the spurious pattern in Figure 1 is -1593.0.

The energy from 100 separate runs from different random patterns as the ones that can be seen in Figure 3. The steady state energy end up on four different levels indicating that there are four different attractors in the network (8 if counting anti-patterns).



Figur 3: Intermediate state energy from 100 different runs such as the one in Figure 2.

When the weight matrix is set to a normal random distribution the energy never settles. An example of this is displayed in Figure 4 where the left image displays the energy from three different starting points and weight matrix. The right image in Figure 4 depicts the energy from 10 different runs with a symmetric weight matrix. The number of runs visualized in the images have been chosen to maximize the aesthetic value.

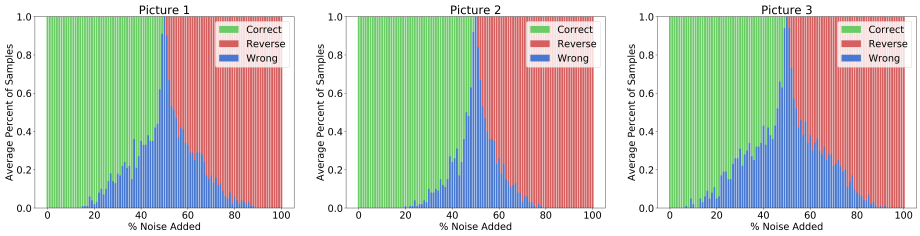


Figur 4: The left image depicts the energy from a normally distributed weight matrix and the right image the energy from a symmetric matrix.

3.4 Distortion resistance

To measure the resistance of the network to distortion, we added noise to each input pattern by flipping the bits in a certain percentage of values. Since the network is able to get the same energy from a pattern and its opposite, we had interesting results when testing the distortion of the network. In summation, over an average of 100 iterations per noise value and up to 1000 iterations, we

discovered that the network performed the worst when given half noise and best when it was given less than 20% noise or more than 80% noise, as can be seen by the graph in Figure 5. You can also see that Picture 2 was the least affected by noise, followed by Picture 1 & 2. We did see some spurious pictures emerge among the wrongly classified pictures (blue bar).



Figur 5: Network recognition with varying amount of noise.

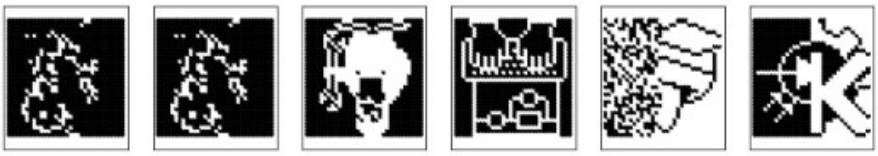
In general, the extra iterations didn't seem to greatly affect the performance of the network. We compared the single-step Little recall to iterations of 100, 250, 500, and of course 1000 (Figure 5). There weren't any differences that were large enough to be unattributed as a sample size flaw. However, that is not to say that the more iterations performed worse, they just didn't perform significantly better.

3.5 Capacity

The amount of patterns that could be stored varied, and mainly on how different the patterns were from one another. Based on our observations, we could store from 2 to 5 patterns reliably, depending on which combination of patterns were selected. If three patterns were selected that were similar to each other (i.e: the binary ratio, where they were concentrated, and similarities in patterns), then they would struggle to be stored and recalling would either give a spurious image or incorrect recognition. However, if the patterns selected were sufficiently different from one another, we could store upwards to 5 patterns, though the results of recognizing a distorted one was not reliable and became less accurate as more patterns were stored.

The drop in performance in terms of storage dropped abruptly, as there were also cases where storing an additional pattern would make the network unable to remember the (n-1)th pattern, where n is new pattern, as in; if we made it store 5 patterns, it stored 5. If we added one more pattern to store, so that it would store 6 patterns, it ended up storing only 4 or even 3 in certain scenarios. This is due to the fact that these patterns have converged to a local minimum, hence the spurious patterns that emerge when recalling.

With all of this in mind; the network was able to store random patterns more consistently if they were sufficiently different from the images as to not be confused with each other and not to interfere with the energy in a way that it will lead to convergence of local minimum, though the network became increasingly

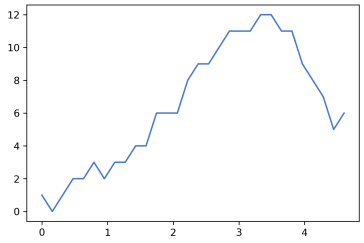


Figur 6: Learning a sixth pattern made the network unable to recognize the fifth pattern which was previously successfully stored (read right to left).

more fragile as more patterns were stored of any kind. This was especially the case with a larger network, as the network was a lot more prone to falsely recognizing patterns, most likely due to the trained patterns being similar enough that they would end up intruding upon each other. With a larger network, we could store upwards 15 patterns or more, though it depended on the factors mentioned above.

3.6 Sparse patterns

For the last question, we generated sparse patterns and varied the value of the bias term from 0-4.5 (with increments of .15) for an activity of 10%. Figure 7 shows our findings, with the bias of 3.45-3.6 storing the most of patterns (12).



Figur 7: Numbers of patterns with 100 features learned based on bias term (activity = 10%).

After computing the activity at 10%, we carried a similar procedure out for an activity of 5% and 1%, however we changed the bias terms to 10 evenly spaced terms between 0-17.5. At 5% activity with 500 features, the best bias range was 9.72-13.61 with 98 patterns learned. At 1% activity, the best bias range was (add data here) with (add data here) patterns learned. One issue we ran into with these terms was the that the vector was so sparse that we had to increase the number of features/length of the vector to actually get valid results.

4 Final remarks

We found it very interesting to see the Hopfield network work in practice, especially with the visualization in task 3.2.

The assignments and the nomenclature in the assignment were not always clear to us.

It is never specified if sequential updating is the same as asynchronous.

It is never clearly specified what is meant by iterations even though the this can be interpreted differently with synchronous and asynchronous updating.

It is also not very clear how we are supposed to work with the weights on the main diagonal as they are not mentioned in the background.

It is also unclear what single-step recall is supposed to mean.